EMGT 835 FIELD PROJECT

Web-Based Software Vendor Management Application

By

Naresh Puppala

Master of Science

The University of Kansas

Spring Semester, 2010

An EMGT field project report submitted to the Engineering Management Program and the faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

_____

Name: Dr. Tom Bowlin
Committee Chair

_____

Name: Dr. Raymond Dick
Committee Member

_____

Name: Ejaz Malik
Committee Member

_____

Date Accepted

Table of Contents

List of Figures

List of Tables

Acknowledgments

Thank you to my wife, Deepthi, and my son, Pranav; their patience with me while I was working on this research was incredible. They supported and commiserated with me during the ups and downs that came during my studies at the University of Kansas. Love you both!

I am thankful for my Field Project committee chair, Dr. Tom Bowlin, and for his insight and guidance for my project. I would like to extend special thanks to my project committee, Dr. Tom Bowlin, Dr. Raymond Dick, and Ejaz Malik, for their comments and helpful suggestions. I appreciate their extra time and commitment to support my academic success.

The research would not have been started were it not for the initial suggestions of Annette Tetmeyer. Her persistence, encouragement, and patience have helped shepherd this project through many months of creation, modification, evaluation, and production. Thanks also to Lisa Lord and Joy Bancroft, who helped me through the researching and writing process. I could not forget to thank Venkata Yerrapothu and Raghava Ankem at Yash Technologies, Moline, IL, who have provided valuable suggestions and advice on technical subject matter. Thanks also to Denise Karimi for her collaboration during data collection.

I would like to thank Parveen Mozaffar for her support for all these years. Thanks finally to the faculty, fellow students, library, writing center, and administration, who have all contributed to my learning experience at the University of Kansas.

Executive Summary

At XYZ Technologies, Inc., the lack of standardized processes and tools to manage different software vendors is costing the company time and money. Visibility and the ability to track software vendor transactions are limited due to the lack of a centralized vendor tool repository, resulting in duplicated software throughout the company. In addition, tools are not being used to track vendor performance, which prevents the company from both negotiating critical service level agreements as well as monitoring vendor conformance to service level agreements.

Based on research regarding software vendor management, a web-based application has been developed that would provide support for vendor trouble tickets, license utilization, and performance metrics. All the software used in the development of the application is available free of cost. The application was developed based on software development concepts of maintainability, reusability, and extensibility.

The developed application would streamline the vendor trouble ticketing process and improve communication between vendors and development teams. Vendor performance metrics help management make decisions during vendor software contract renewals. The newly developed centralized tool would reduce costs by better managing software licenses, reducing time to resolve issues, and providing for accountability of software vendors.

1 Introduction

At XYZ Technologies, Inc., the lack of standardized processes and tools to manage different software vendors is costing the company time and money. Currently, vendor data is scattered throughout multiple applications that are only accessible to the supply chain manager. Other stakeholders like developers, architects, and project managers have limited access to this data. One vendor can be managed by multiple account managers, and therefore, vendors have multiple contracts, sometimes for the same software.

The lack of a centralized vendor tool repository has caused multiple teams in the same organization to purchase different software packages to perform the same business function, which results in duplicated software throughout the company. Tools are not being used to track vendor performance, which prevents the company from both negotiating critical service level agreements as well as monitoring vendor conformance to service level agreements (SLAs). Therefore, when a vendor fails to deliver products or fixes on time, the company cannot readily impose penalties. Contract agreements are not clearly defined, and the company incurs additional costs for software enhancements, training, trials, prototyping, and upgrades.

The research objective was to bring transparency to each transaction related to software vendors. Research was conducted to examine how vendor licensing information, ticket resolution, and other vendor data are stored and how accessible the data is to non-supply chain management teams. Currently, only the executive team is involved in the selection of vendor software and licensing. During usage and software implementation, business and development teams are facing problems with the vendor and its software, both

technically and with support; the root cause is little to no involvement of the technical team in the vendor selection process.

As a result of the research findings, a web-based application has been developed for software vendor management at XYZ, Inc. The vendor management application would share XYZ's priorities with its vendors so that vendors would know exactly what is expected from them. XYZ could then build business cases for new software purchases based on information from the vendor's white papers and by using financial analysis tools such as return on investment. Vendors would benefit from the newly developed application by having a better understanding of the organization's needs and being able to refine their proposals to meet these needs. To achieve better results, both the vendor and client must act like partners; when a software vendor commits to a result rather than a sale or business transaction, its success is directly tied to XYZ's organizational success.

The developed application would eliminate existing problems and improve current processes. The application would track trouble tickets, licensing, and service level agreements. Vendor performance can be measured and evaluated based on the number of software fixes per year, trouble ticket response, problem resolution, and the number of developer-to-vendor calls to report software issues.

2 Literature Review

A vendor management system (VMS) is a web-based application that businesses use to procure and manage vendor software and services. Typical features of a VMS application include vendor selection, process and planning, troubleshooting, and license and contract management. Quality is also a major consideration when developing software applications. This chapter addresses two main areas: the selection of vendor software and quality software standards for building a new web-based application for software vendor management.

2.1 Vendor Management Software Selection

An extensive literature review was conducted to determine how to select the best vendor management system. Six key elements are as follows:

- General process and planning
- Software vendor selection tips
- Questions to ask prospective vendors
- Vendor software security
- Vendor management office

Each of these elements is discussed in the following sections.

2.1.1 General Process and Planning

To remain in touch with emerging markets and changing economies, vendor management software should be updated to save both time and valuable resources spent accumulating the data provided by the current software. Therefore, it is important to understand that the selection of vendor software is a business decision as well as an IT decision. The main question in vendor management discussions is whether to develop software in-house or to pay for an external vendor. Before making the decision, it is best to understand software licenses, prices for hardware, software, and support. In many cases it often makes more sense to use vendor products that are specialized for a specific use.

Vendor software affects not just the technical aspects of the IT infrastructure, but also the business processes used by employees as they conduct their business. The vendor selection process is not an isolated process that is limited solely to software task-related activities; its scope is quite broad and complex. The actual task of buying involves many people and addresses numerous issues, so decision processes regarding vendor software selection can take many months and engage employees from all parts of a business. The end users' involvement at the beginning of the software selection process is a key success factor (Verville & Halingten, 2001).

In today's competitive marketplace, information is critical to business operations. Managing information requires various applications to satisfy business needs. These applications need to be compatible with each other, which is challenging given the wide range of operating systems and platforms across the enterprise. Operating system compatibility greatly impedes the selection of vendor software or system(s).

There are significant cost factors that also need to be considered. Software from vendors is a high expenditure that requires a significant portion of an organization's capital budget. Critical to an organization's success is the replacement of an out-of-date system with the selection of vendor software with the lowest price or latest technology. Expenditures for vendor software packages can range from a few hundred to even millions of dollars. Therefore, it is critical to have tools that take into account both system requirements and cost to make the best decision when purchasing software.

A wrong purchase can adversely affect the organization as a whole in several different areas and on different levels, even to the point of jeopardizing the existence of the business. For example, the advanced baggage handling system at the new Denver International Airport has become one of the most notorious examples of a software project failure. The problems building the complex baggage handling system resulted in the newly completed airport sitting idle for sixteen months while engineers worked on fixing the baggage system. Approximately $560M was added to the cost of the airport (Calleam, 2008). Many software project implementations have failed from the start because organizations simply chose the wrong software based on the "bandwagon effect" or "because so and so bought it" (Verville & Halingten, 2001). This choice results in a gap between the organization's needs and the software's capabilities. Waiting until the software implementation to consider needs such as cost effectiveness, the fit of the software, etc., is simply waiting until it is too late and inviting failure. Attention to software drawbacks at the time of the acquisition would subsequently reduce or eliminate delays and cost overruns during the software's implementation.

According to Marciniak and Donald (1990), the software acquisition life cycle contains three major phases: concept exploration, source acquisition, and performance management. During concept exploration, the buyer prepares functional and technical specifications and develops an acquisition management strategy. During source selection, a vendor is chosen to develop the system or provide software/hardware based on the vendor's proposal. In performance management, the buyer monitors the vendor's performance and compliance with contract terms.

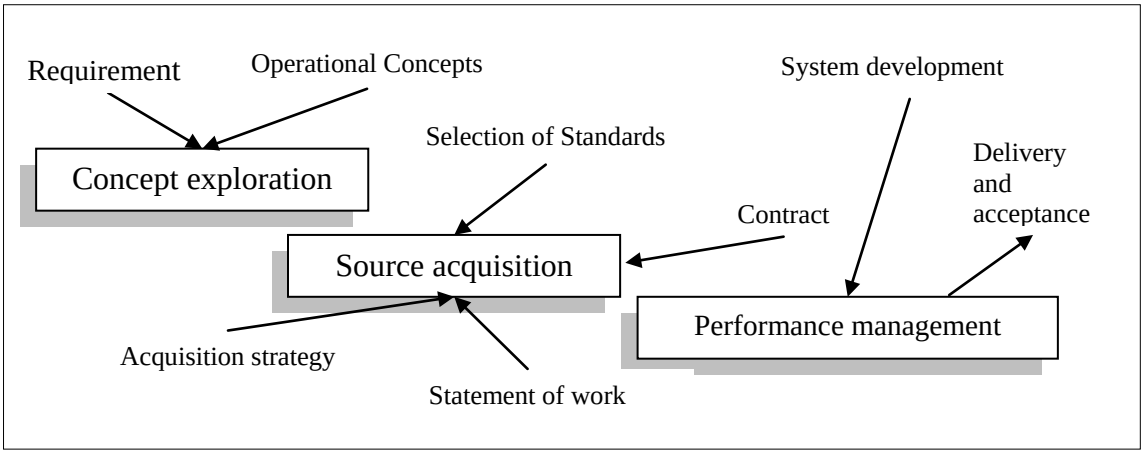


Figure 1: The Acquisition Life Cycle
(Marciniak & Reifer, 1990)

Verville and Halingten (2001) cite several factors that are critical to the successful implementation of vendor software, including:

- Strong leadership and management commitment
- Careful selection of the acquisition team members
- Planning – an absolute must if the acquisition is to be success
- Definition of the requirements – all requirements should be on hand prior to looking at vendor solutions

- Selection and evaluation criteria – an absolute must prior to meeting vendors
- Functional and technical evaluation of vendors
- User involvement and user buy-in of final choice

Each organization has a software acquisition process. The type of technology being sought, individuals and departments involved, and their influence will conspire to form the overall nature of process. The acquisition process will change slightly depending on the solution that is desired, the impact on internal users / customers, the culture of the department, and the funding approval process. This process is inter-related and iterative and will have the following sub-processes (Verville & Halingten, 2001):

1. Planning
2. Information selection
3. Selection and evaluation of vendors
4. Negotiation

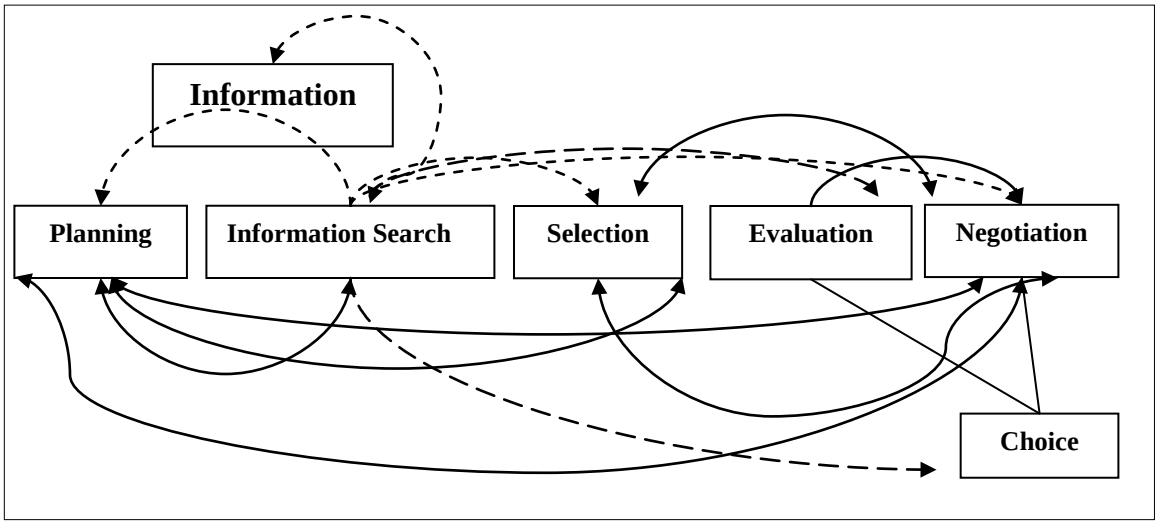


Figure 2: The Enterprise Software Acquisition Process
(Verville & Halingten, 2001)

From the author's experience, the planning process cited in various sources is similar for many organizations. Typically, the planning process begins shortly after the decision is made to purchase vendor software. An acquisition team is formed, and during initial meetings the information search process begins. Requirements are gathered, selection criteria are established, and then the evaluation process begins. The acquisition team establishes strategies and a time frame to monitor issues that are related to the acquisition. During market analysis, information on vendors and their solutions will be screened based on functional and technical criteria. A short list of vendors and their solutions are chosen that match the organization's requirements. The next step is to prepare the Request for Proposal (RFP) and send it to the short-listed vendors.

Figure 3 shows the acquisition planning process that contains the seven categories of major activities that should occur during this process.

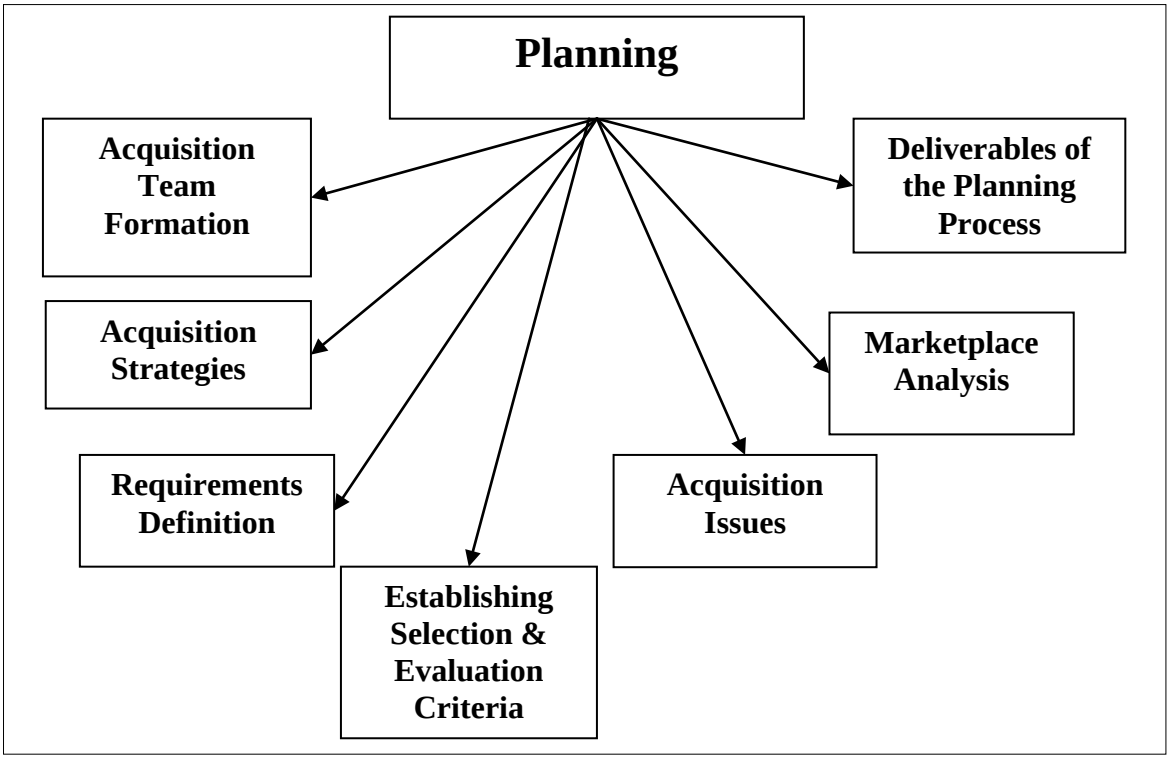


Figure 3: Planning Process
(Verville & Halingten, 2001)

2.1.2 Software Vendor Selection Tips

Vendor management software varies widely depending on the services a firm offers, as well as the specific needs that the company has. The VMS must work with the company's existing operating system or the company will have to upgrade the entire system to fit the software. The software must be network-ready for multiple sharing over departments and branches.

Many software systems will contain information that may not be required for all firms; the company should not pay for options that do not work for it and should select a software system that will grow with the business. Alternatively, upgrading the company's present software with new functions may be the best option if the system in place is already working. Carver (2009) advises when choosing vendor management software, a company should consider these features:

"Purchase the VMS that is right for the company"

Locate online VMS providers with programs that can be used by the firm. Check the features of the software that is being considered and purchase what the company wants or what is needed. The software should effectively organize all vendor information in one place so that the necessary information can be easily accessed and reviewed. All vendor contracts should be accessible from one software system / service.

“Work with vendor management solutions that are already in use”

If the present software is working, consider upgrading it to add the few features that are needed instead of replacing the entire system to save costs. Alternatively, choose a software program that adds these features but integrates them with the program in place.

“Find help with the VMS through customer service support”

Even the best software is of no use if someone cannot be reached to answer questions when the need arises. Choose a software system with live customer support, such as help phones, email, or online chat, to get the most out of the program. For non-urgent questions, an online form would be helpful.

Lisa C. Ray (2009) cites a number of ways VMS can be integrated into operations. Each of these is discussed below:

“Try various VMS providers”

Check with peers for referrals and investigate VMS sellers. Many vendors provide a free trial period; it is a good idea to request a minimal trial time before signing any long-term agreements to make sure the software is the right program for the organization’s needs.

“Hire a VMS system specialist”

Invest in an outside consultant to review the company’s needs and to make recommendations for the best vendor management software. Consultants have had

experience with various vendors and often know which software is performing the best.

“Outsource company operations to suppliers of vendor management software”

Reduce costs by utilizing the services of a vendor management software seller that also can contract for an outsourcing option. Turn over the IT completely to a VMS provider that can set up new systems and manage the day-to-day operations. Make sure the consultants have a help line available 24/7 for client needs and that the vendor will be quick to respond to issues.

According to Joseph P. Savidge (2008), “Selecting the right vendors and properly managing vendor relationships can help protect a company from damages and long term losses.” He cites six major points that play a key role in vendor software selection and utilization:

“Invite many vendors to participate in the RFP”

Prepare an RFP that covers the majority of business needs while setting expectations for the vendor from all perspectives (legal, security, etc.). Vendors with the most flexible terms may be given priority for contract negotiations.

“Prepare a detailed questionnaire”

To check the vendor’s capacity, operations, policies, and security, provide a detailed questionnaire. Use the responses to gauge the vendor’s compliance with policies important to the company.

“Review vendor’s audited financial statements”

Understand vendors’ levels of research and development spending on their products. Periodically check vendors’ financial statements and their overall ratings.

“Ensure the contract terms are beneficial to the company”

Review contracts for penalties, termination clauses, warranties offered, and maintenance fees charged by vendor. Ensure there is a return of proprietary data at termination, remedies for breaches, and conversion assistance at termination. The SLA should benefit the company and be measurable and enforceable.

“Track vendor performance and compliance”

Periodically check with a recipient of the vendor’s services to assess the vendor’s performance. There should be internal discussions regarding vendor performance. If the vendor does not meet the contractual agreement then changes should be made to the contract.

“Maintain an inventory of contracts and licenses”

The vendor’s initial contracts, current contract amendments, license information, dates of notification, renewals or termination, and total annual value of contract details should be kept updated.

Other authors also agree on key aspects for selection of vendors. According to Cullmann and Goldstein (2003) to achieve a company’s business goal, four main areas should be considered in the vendor-selection process:

- The financial and management wherewithal of the vendor
- The vendor's track record and reputation
- The vendor's expertise and capabilities
- The internal-control environment of the vendor

Financial matters affect the performance and existence of a vendor. To avoid this scenario, the company should conduct sufficient due diligence of a vendor's financial performance and future prospects to be able to assess the vendor's staying power in business. In addition to analyzing a vendor's financial statements, conduct credits checks and verify the identity and integrity of vendor's key management team.

The client should understand what experience the vendor and its personnel have in delivering the service or support. One way to verify experience is to check with the vendor's clients who have contracted for service to gauge reliability and satisfaction. It is imperative that a vendor has enough capacity to support and protect the interests of its clients. Companies should understand and plan for the impact to their business in case of a vendor service's failure. The vendor's backup planning also must be reviewed periodically. It is important that the client develop an internal process to monitor the vendor's service and performance quarterly, semiannually, or annually.

Vendor performance results should be shared and discussed with the vendor for improvements if needed. A client's audit should review performance, support, and high risks from all of its vendors (Cullmann & Goldstein, 2003).

2.1.3 Questions to Ask Prospective Vendors

It is very important for a prospective customer to check for specific questions related to the particular area of the vendor's specialty and the organization's vision and business needs. Healthcare Financial Management (2009) cites a list of questions to ask vendors, including:

- What is the vendor's track record with customers over time?
- What kinds of references can the vendor provide?
- What are the vendor's main points of differentiation with competitors in its sphere?
- What are the vendor's mechanisms for interfacing its products with other software and applications?
- What kinds of customer support does the vendor offer?
- What type of delivery options can the vendor provide?
- What kinds of data does the vendor have on long term maintenance, support, and infrastructure costs related to its products?
- What is the vendor's strategic vision of its product marketplace?

To get better results, the IT personnel from the company and vendor should understand the importance of communication and cooperation. Trust between the parties influences the vendor's software usability. Trust is associated with the vendor's dependability and responsiveness.

2.1.4 Vendor Software Security

Clients should be careful about a product's "back-door" access methods; some companies insist vendors sign a contract that their products do not have back-door access methods. Some companies require an independent code inspection to ensure that back door security access or other potential security threats with code are not included in the vendor products. At the same time, companies acquiring the software should insist that their resources follow coding standards and use strong username and password combinations.

2.1.5 Vendor Management Office

A better approach to deal with vendors is to aggressively evaluate their ability to add value to an organization's business. Geoff Koch (2007) writes that companies handling more complex IT offerings and dealing with multiple vendors are increasingly forming Vendor Management Offices (VMOs) in their IT departments, which oversee RFPs, negotiations, contracts, and maintenance of relationships with vendors. Ideal members of a VMO are the contract executive, vendor manger, financial analyst, performance analyst, and administrative support (Koch, 2007). According to Leo Lingham (2009), roles and responsibilities in a VMO are described as:

Contract Executive: A member of senior management; oversees the vendor contract and is ultimately responsible for the program's success.

Vendor Manager: Under the direction of the contract executive; works closely with vendors and is responsible for the quality and cost effectiveness of vendors' services.

Financial Analyst: Provides financial analysis and audit work in support of service contracts and reports to the vendor manager.

Performance Analyst: Analyzes vendor performance in order to assure compliance with service-level agreements and continuous improvements of services.

Administrative Support: Ensures that all changes to contracts are made in accordance with base agreements and interests of the company.

The financial analyst, performance analyst, and administrative support directly report to the vendor manager (Lingham, 2009)

2.2 Quality Principles

Quality software means more than just a program that runs without any errors or without locking up the computer software or hardware. The main purpose of any software project is to be efficient and reliable. According to Dale, Joyce, and Weems (Nell, Daniel, & Chip, 2006), there are four quality goals that all software should meet to some degree:

- It works
- It can be modified without excessive time and effort
- It is reusable
- It is completed on time and within budget

Any tool or application developed to resolve the vendor management issues should satisfy these four quality principles. Features and tools of VMS should meet all the requirements, such as consolidating vendors, troubleshooting vendor tickets, reporting, and outperforming manual systems and processes.

According to the Shewhart cycle (Aguayo, 1991), Plan-Do-Check-Act (PDCA) is an iterative four step problem-solving process typically used in business process or quality improvement. Figure 4 represents the Shewhart process improvement cycle.

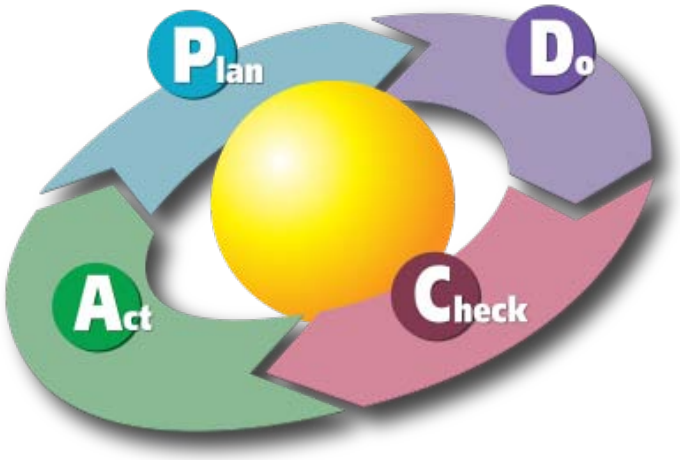


Figure 4: Plan-Do-Check-Act Cycle

(Bulsuk, 2009)

In the Do part of the PDCA cycle, a vendor and related application must be selected for real time VMS implementation. Once the VMS is implemented for a trial period, feedback should be collected from both the vendor and users. Feedback should be checked to determine what does and does not work with the new VMS application. In the Act phase, necessary changes must be made to the web-based application based on feedback. Once changes are implemented, all stakeholders must be tested again for further changes. The capacity of the hardware and network should also be tested during this process. At this time, an additional cost analysis should be done to check whether or not it is worth the development of a new web-based application in house. When comfortable, the VMS application should be deployed on a large scale with all current vendors. Once enterprise-wide deployment is completed, new features should be added and changes made based on stakeholder requests.

3 Research Procedure

The research procedure involves three phases: analysis of the present situation, analysis of commercial-off-the-shelf (COTS) vendor software, and approach of software development.

3.1 Analysis of the Present Situation

Currently, XYZ, Inc., lacks a tool to manage software vendors. A key concern due to the lack of a vendor management tool is that vendor licenses are underutilized; XYZ is overpaying for vendor licenses by purchasing additional licenses instead of simply utilizing unused licenses from a vendor. Figure 5 and Table 1 show the difference between the total purchased licenses and actual license usage for the year 2008. In some cases, fifty percent of licenses may be underutilized from a single vendor.

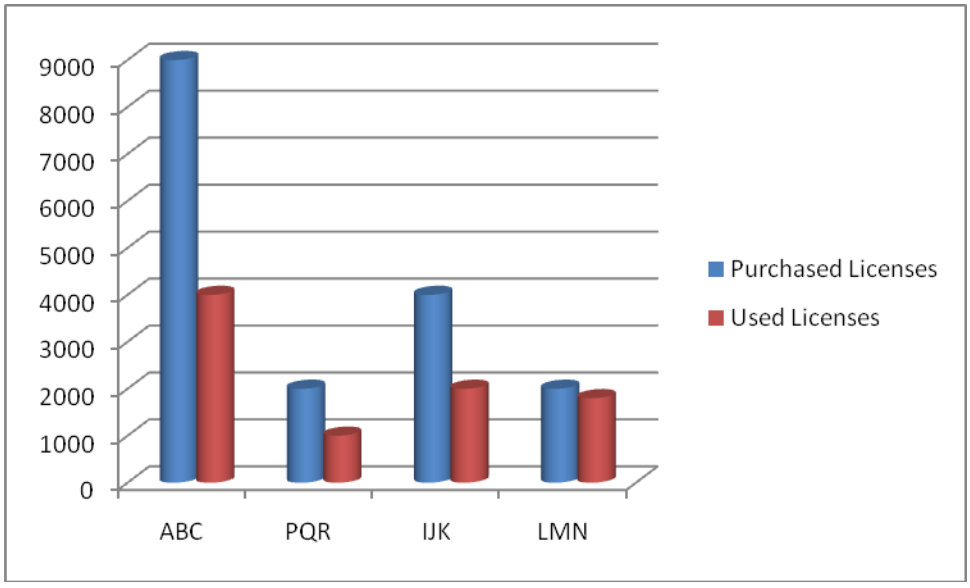


Figure 5: Vendor Licenses Data for Year 2008 (Estimated)

Vendor	Purchased Licenses	Used Licenses
ABC	9000	4000
PQR	2000	1000
IJK	4000	2000
Other	2000	1800
Total:	**17000**	**8800**

Table 1: Vendor Licenses Data for Year 2008 (Estimated)

In addition to the problem of underutilized licenses, there is large variation in vendor costs from the time of initial contract to execution time, which results in significant additional costs. Table 2 shows estimated vendor contract initial and additional costs for the year 2008.

Vendor	Contract Initial Cost	Additional Costs
ABC	3	0.5
PQR	5	2
IJK	6	3
LMN	1	0.025

Table 2: Vendor Costs (in millions)

Without vendor performance metrics, the company is unable to impose fines on vendors when they fail to fix problems reported by trouble tickets within the contract agreement's time or to provide solutions to software problems. Table 3 represents the number of tickets created with vendor ABC in the last four years.

Year	Number of Tickets	Unresolved Tickets
2005	8	1
2006	5	0
2007	1	2
2008	10	2

Table 3: Vendor Trouble Tickets Data

Therefore, lack of a vendor management tool results in difficulty in tracking underutilized licenses, inability to easily track post-contract additional costs, and inability to track vendor performance metrics. The research focused primarily on the vendor

trouble ticketing process. License tracking and vendor performance metrics are available but were not the main focus for the research.

3.2 Analysis of COTS Vendor Software

Many COTS vendor software applications are available in the market. A partial listing of software readily available in the market for vendor and supplier management includes:

- Vendor Organizer Deluxe v3.2 (Primasoft pc software)
- EC Sourcing Group Vendor Management Solutions (EC Sourcing Group)
- Vendor Risk (Skeey Interactive LLC)
- Vendor Management Software (Beeline)

All of these products have common features and, in addition to being web-based, include the abilities to: customize; create a vendor database easily; search and replace; generate vendor reports; modify data; backup database easily; filter and sort vendor records; and produce summaries, graphs, and statistics. However, most of the systems do not include integrated vendor trouble ticketing support. Based on research, integrated vendor features are a major element for the proposed web-application.

3.3 Software Development Approach

The most important task in creating a software application is collecting requirements and then analyzing these requirements. Incomplete, ambiguous, and contradictory requirements should be addressed before starting the planning process. Once general requirements are finalized, then the scope of the project is determined. The application

was developed using activities from the software development process represented in the waterfall model.

Figure 6 illustrates the different phases in the software development life cycle using the waterfall development approach.

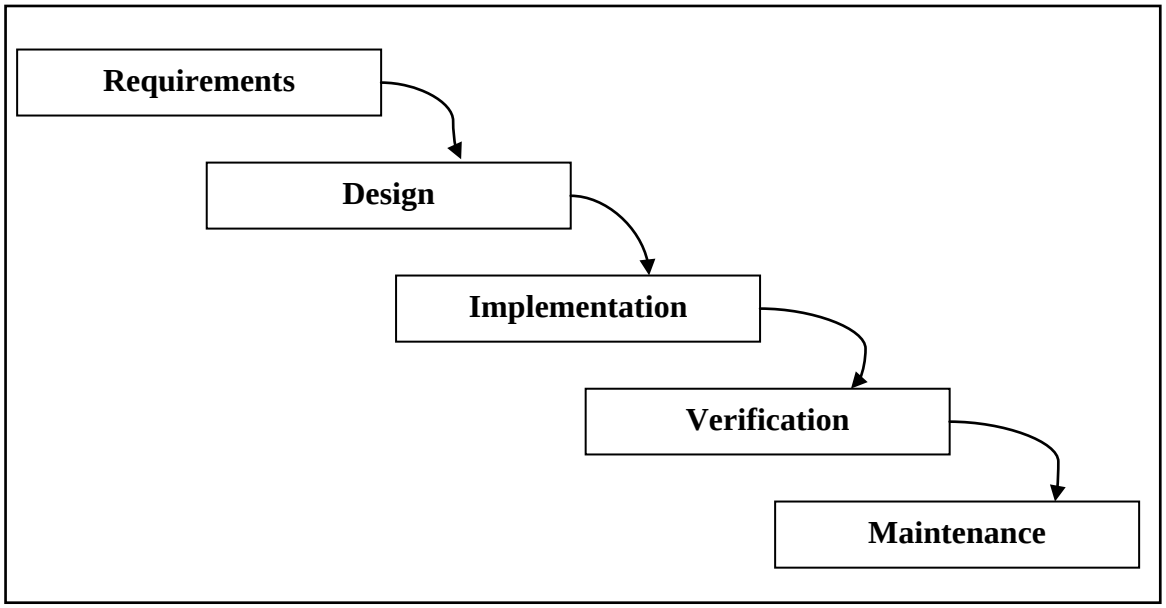


Figure 6: Waterfall Model

By developing the application phase by phase and reviewing at each phase before transitioning to the next, numerous changes in coding and regression system testing can be completed to better meet the project requirements. Development of the web-based application followed a generalized waterfall model.

4 Results

The new web-based application for vendor management is intended to support vendor trouble tickets, license utilization, and performance metrics. This chapter covers the vendor trouble ticketing process, application details, database design, application features, quality of the VMS application development, and application advantages.

4.1 Overview of the Trouble Ticketing Process

The main goal of the developed web-based application was to monitor tickets created by developers for vendors. Currently, when developers create tickets for problems or patch-related info for vendor software, they need to login to the vendor's website by using valid login credentials, sending an email, or making a phone call to the vendor's contacts / customer service to create a ticket. The problem is that the developer has no centralized data storage for ticket info. This creates a situation where two or more developers from the same company but from different organizations or teams can create tickets for the same or similar problems. In addition to creating redundant work, some vendors charge clients on a per ticket basis. Trouble ticket duplication is costly.

For the developed solution, each vendor's software was divided into products, which were subdivided into applications. Access to the VMS application was designed based on the Role Based Access Control concept, which is used in an IT systems access control management. Based on the user's role, two groups are created: user and administrator. Users can search existing tickets using different search criteria, create tickets for vendors, and update their tickets. Administrator roles are created for subject matter experts, who

mediate between developers and vendors based on their application software. Administrators can search, create, update, and forward a ticket to the vendors if required. Also, the administrators can add or update vendor, product, project, and user information.

When developers have questions related to vendor software, they first need to login to the VMS application using their secure user ID and password. Once logged into the system, they can search the database for existing tickets. If they do not find a solution for their problem, then they are able to create a ticket. Once the ticket is created, it is submitted to the application administrator for that particular vendor. The administrator reviews the ticket and decides whether that ticket is valid or not; if it is valid, the ticket is forwarded to the vendor site using web services. From there, the vendor starts corresponding to the ticket originator and application administrator with a suitable solution and suggestions. For a completed transaction, such as the creation of a ticket or a forwarded ticket to or from the vendor, an email is generated to the ticket originator and administrator for that particular vendor. Whenever a ticket is created, a token from the vendor is generated and stored in the tickets table for future reference and tracking. Figure 7 illustrates the vendor ticket flow in the VMS application.

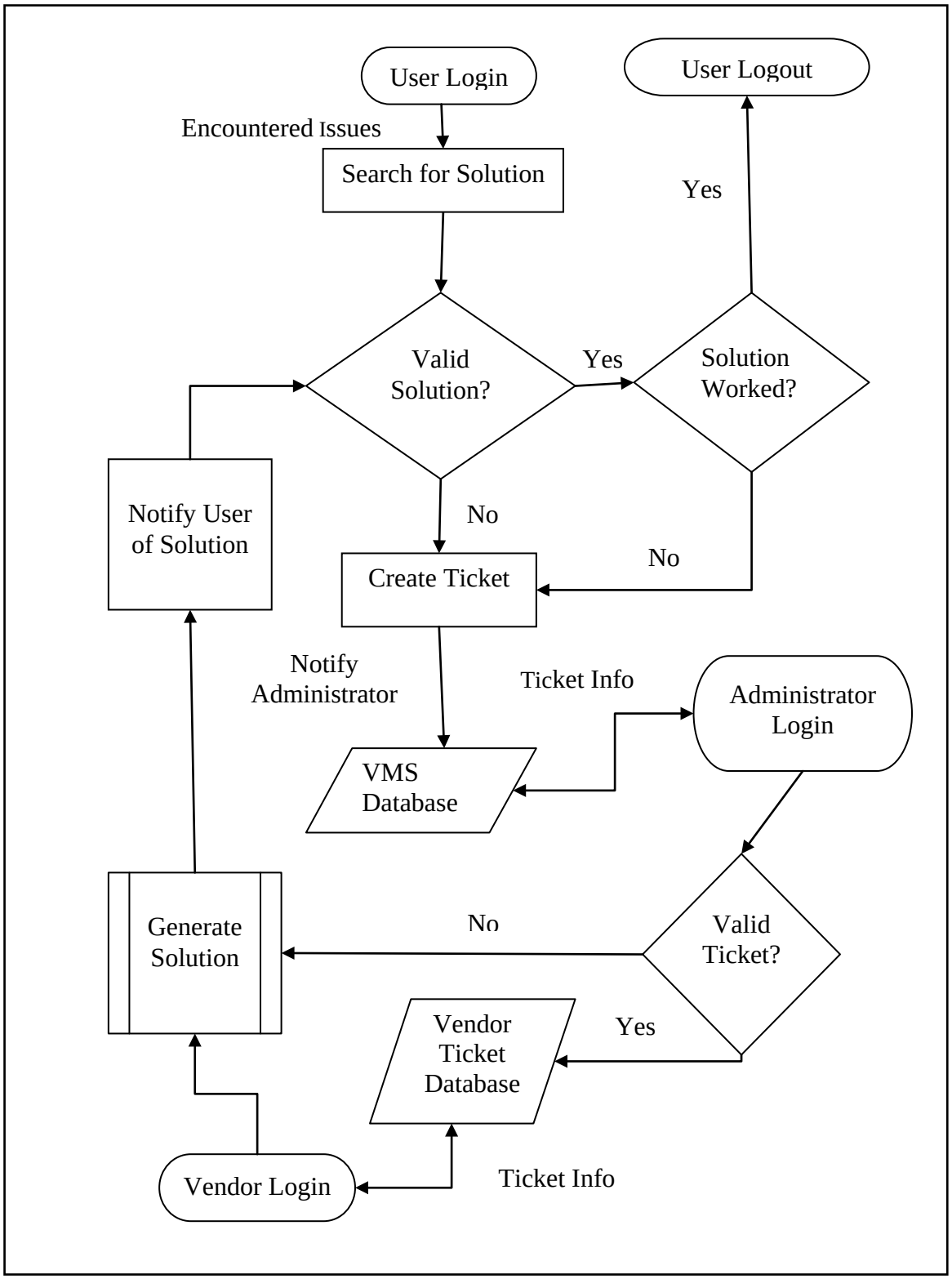


Figure 7: Trouble Ticketing Process

The developed application uses web services to communicate with vendor systems for trouble ticket updates. A web service is “a standardized way of integrating web-based applications. It allows organizations to share data without needing to know the details of other organizations’ computer systems” (Pearlson & Saunders, 2006). With standardized technologies like XML, web services enable applications built on different languages and on different platforms to communicate with each other. By using web services, developers only need to focus on business logic without worrying about execution details. Interoperability and modularity are key features of web services and are excellent for integrating systems across organizational boundaries.

4.2 Application Details

The developed web-based application has three layers: user interface (UI), business (logical), and database. In the UI layer, the user views or enters the data. All the logic is located in the business layer to process user requests from the UI layer. In the database layer, data is processed, stored, or retrieved from the database and provided to the user based on business layer requests. VMS data flow is illustrated in Figure 8.

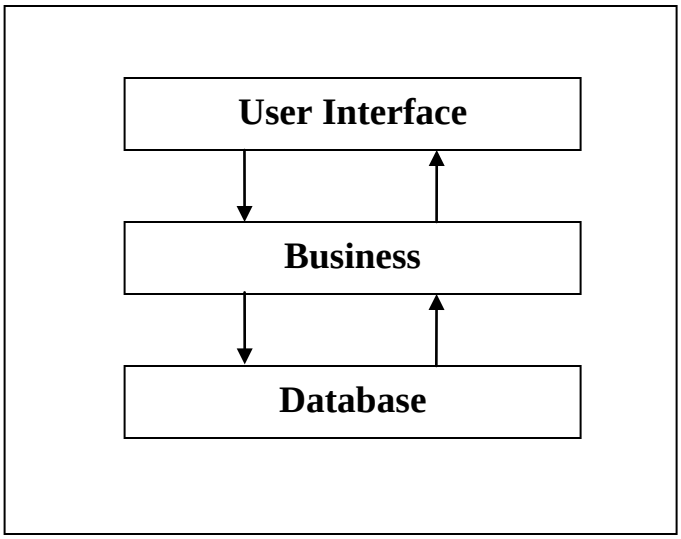


Figure 8: VMS Data Flow

All the software used in the application is either open source or available free of cost when used in applications that conform to GNU not UNIX public license. The licenses for most software for practical works are designed to not be changed or shared. The GNU general public license is intended to guarantee developers' freedom to share and change all versions of a program so that it remains free software for all users. Most of this software is updated by user groups to fix existing bugs. Numerous blogs, seminars, and website support are available for software documentation and troubleshooting. The following is the list of software used to develop the web-based application.

- My SQL 5.1
- Java 5.0
- JQuery and Hibernate framework
- Web services

- Apache Tomcat web server
- HTML
- CSS

4.3 Database Design

For the developed application, the relational database management system was designed carefully and constructed based on a repository of facts and was part of the vendor management system. The VMS database was designed and evaluated within the database life cycle framework. The database provides vendor data collection, storage, and retrieval and also facilitates data transformation. The tables that were created for the VMS schema are: user, ticket, vendor, product, application, project and license. Figure 9 illustrates the entity-relation model for the VMS application database.

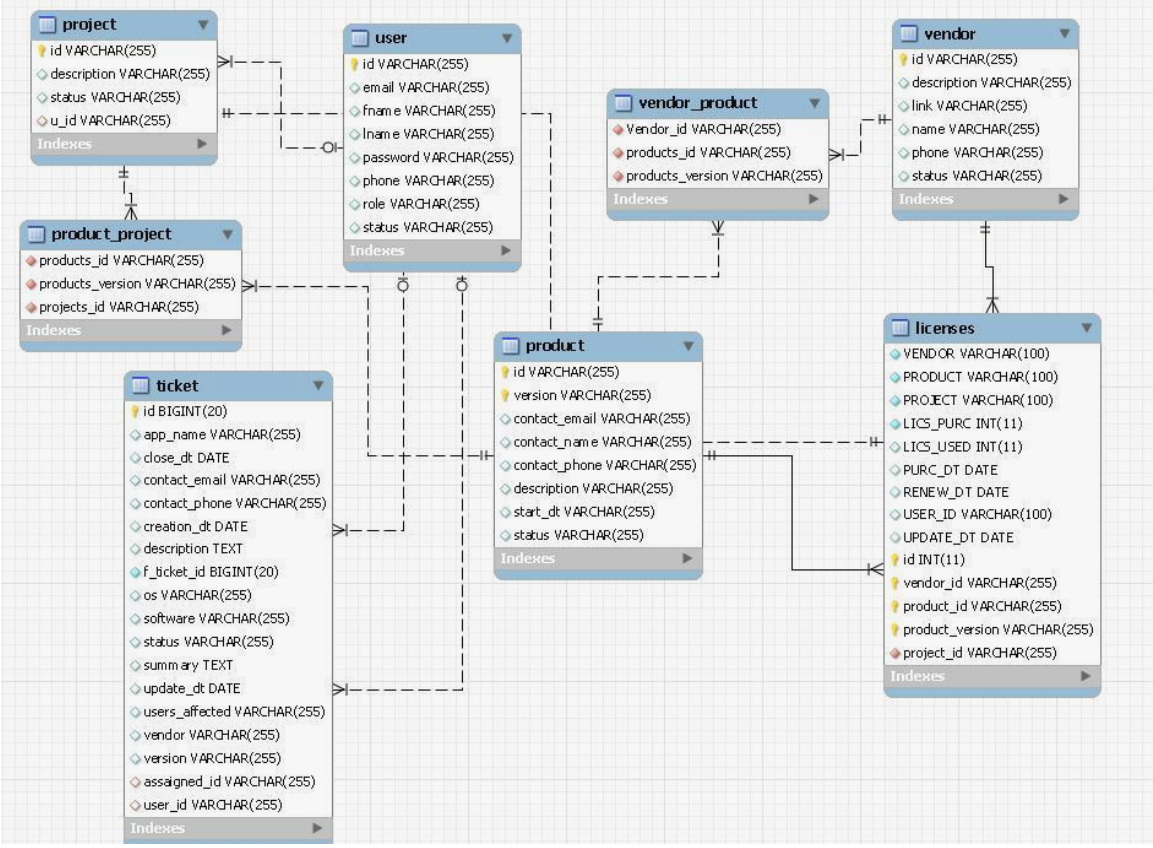


Figure 9: Entity-Relation Model

4.4 Web-Based Application Features

The new web-based application has many features, developed based on present issues with management of vendors at XYZ. New VMS application features include:

Ability to change screen layout: The application gives users the ability to customize the screen layout according to their preferences.

Powerful filters and sort options: Users can search and sort existing vendor trouble tickets using more than fifteen fields in the tickets table (refer to Figure 10).

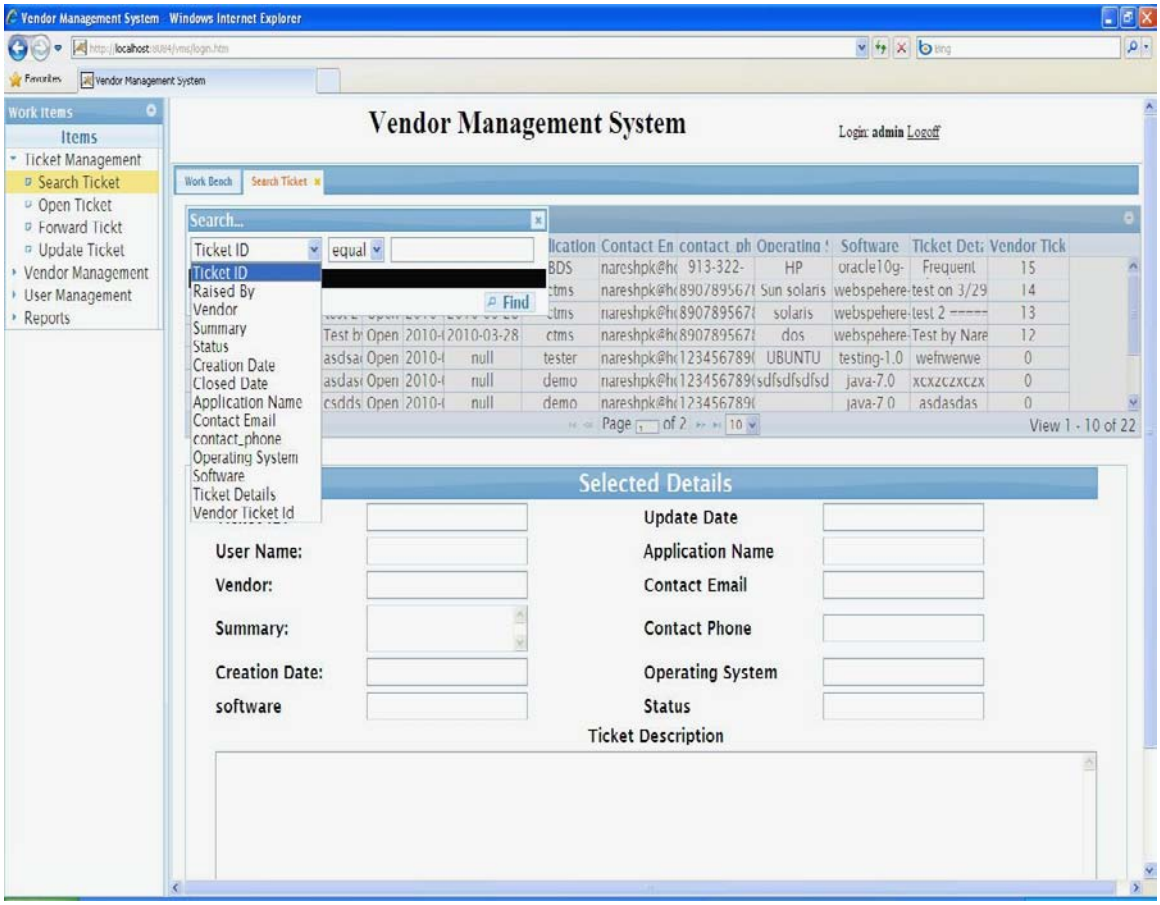


Figure 10: Vendor Trouble Ticket Search Filters

Automatic email notifications facility: The new VMS application generates automatic emails to the administrator(s) and also carbon copies (CCs) the ticket originator whenever a ticket is created, updated, or closed. These email notifications contain ticket details like the vendor name, product name, project name, created by, ticket creator's contact email and phone number, creation date, ticket description, and ticket status. Figure 11 shows an email notification for the creation of a new trouble ticket.

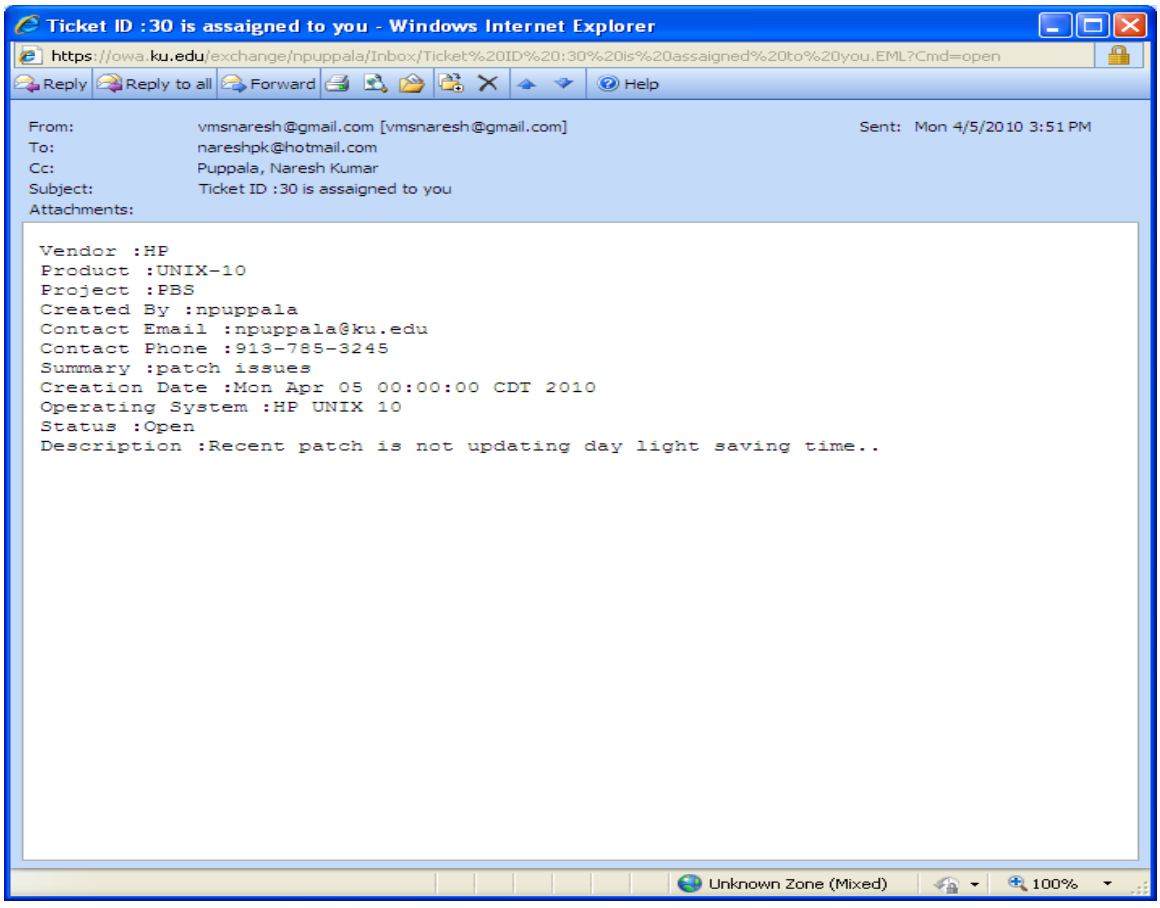


Figure 11: Email Notification

User management: Users can manage their own profiles, and application administrators are able to modify user information such as the password or contact details and can change the status from active to inactive and vice versa based on the user's role (Figure 12).

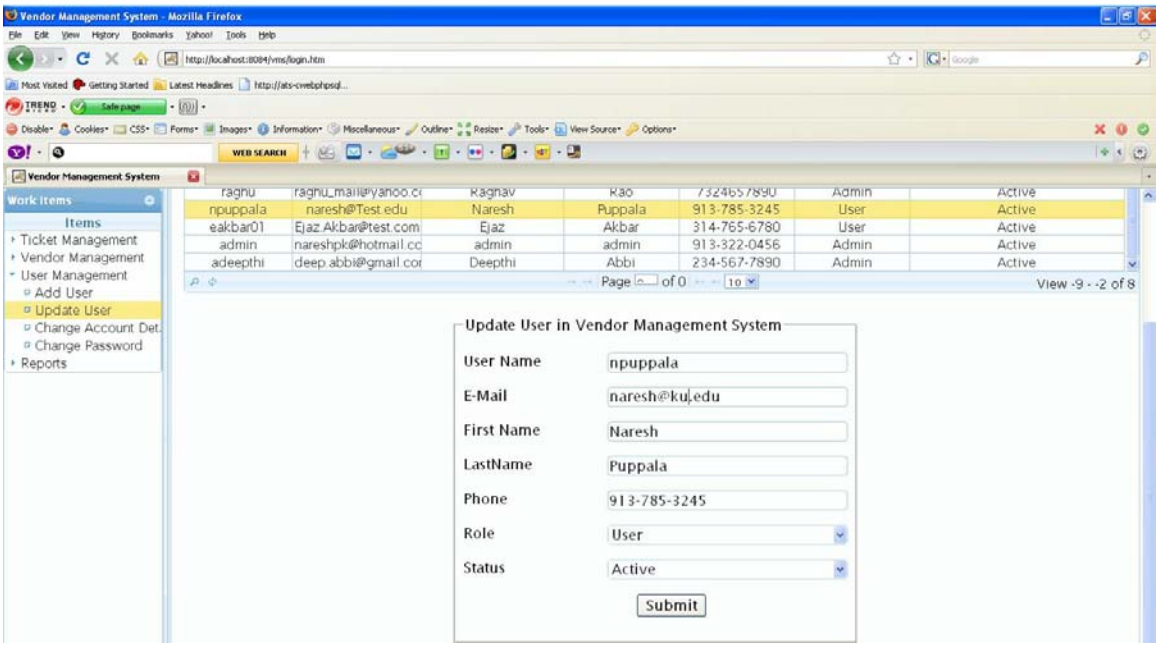


Figure 12: User Management

Reporting: Vendor trouble ticket data, product data, total ticket resolution time, and licensing information can be seen in different formats of graphs and reports (refer to Appendix A).

License management: Each application administrator is able to check vendor license data; based on usage of licenses, administrators can negotiate with vendors whether to increase or decrease the number of licenses (refer to Appendix A).

Auditing: Application administrators are able to monitor trouble tickets created for each vendor, check the vendor's performance, and view vendor licensing usage and license renewals.

4.5 Quality

4.5.1 Quality Software Works

The application provides solutions to issues like handling vendor trouble tickets, licenses information, and communication between vendor(s) and developers. The new application is intended to be user friendly and efficient in performance.

4.5.2 Quality Software Can Be Modified

The web-based VMS application was designed and developed using software development "best practices" so that programs are easy to understand and modify, with each step documented for future changes. Programs developed for the application were designed to withstand future changes, and most of these programs were developed to be relatively independent. Hibernate concepts were incorporated, making it is easy to change the underlying database from open source MySQL to other relational databases such as Oracle, SQL Server, MS Access, etc.

4.5.3 Quality Software Is Reusable

The object-oriented language Java was chosen due to platform portability, making it easy to be plugged into other systems. Application programs were modularized by breaking

them into procedures and subprograms, which improved maintainability, reusability and extensibility.

4.5.4 Quality Software Is Completed On Time and Within Budget

The application was developed using open source language (Java) and database (MySQL), which are available free of cost. The time restriction for the research and the web-based application development was planned for six months. The budget was limited to use open source software. Both timeframe and budget constraints were met.

4.6 Advantages

There are many advantages to using the newly developed web-based vendor management application:

- The new application has been developed to be user friendly; navigation is easy even for new users. Minimal training is required; once trained, any user is able to create vendor trouble tickets, view ticket status, and view vendor information.
- Once the VMS application is implemented, it will be a centralized tool across the company for vendor management and will eliminate all existing tools currently used for vendor management.
- The VMS web-based application will streamline the vendor trouble ticketing process and will improve communication between vendors and development teams.
- Up-to-date data in the VMS application will improve the technical teams' participation in the selection and purchasing process of new vendor software.

- The new application provides metrics on vendor data which can be generated in the form of graphs and charts.
- Vendor performance metrics help management make decisions during vendor software contract renewals.
- With the new VMS application, utilization of vendor licenses and contractual agreements can be monitored.
- The web-based application was designed and developed to integrate well with all vendors.
- Since the new application can be developed in house using the organization's resources, there will be no additional cost or impact on the organization's budget.
- The web-based application allows users to do complex searches and sorting on trouble tickets and vendor data.

However, there were also certain limitations for the developed application. These will be addressed in Chapter 5.

5 Suggestions for Additional Work

A list of suggestions for future developments to the VMS application follows:

- Reports can be generated based on the average time for each trouble ticket for each vendor.
- License usage updates should be automated.
- Use of open-source software should be maintained.
- User customized settings should be saved in database or cookies, and these settings should apply whenever the user logs into the VMS application.
- Projects should be updatable.
- Future changes and enhancements should be based on iterative development such as agile software development methodology.
- VMS application license data and functionality should be capable to support major vendor’s merges.

References

Aguayo, R. (1991). *Dr. Deming The American Who Taught the Japanese About Quality.* (First ed.). New York City, NY: Simmon & Schuster.

Bulsuk, G. K. (2009). "Taking the First Step with PDCA." Retrieved 10 December, 2009, from http://karnbulsuk.blogspot.com/2009/02/taking-first-step withpdca.html#axzz0kO1U2mIG

Calleam, C. (2008). "Denver Airport Baggage Handling System Case Study." *Why Technology Projects Fail.* Retrieved 20 February, 2010, from http://calleam.com/WTPF/wp-content/uploads/articles/DIABaggage.pdf

Carver, L. (2009). "Use Vendor Management Software to Get the Edge for Your Business." *Guide to Knowledge Management Software: Vendor Management.* Retrieved 25 January, 2010, from http://www.business.com/directory/management/knowledge_management/ software/m/vendor-management/

Cullmann, D., & Goldstein, B. (2003). "Effective Vendor Management." *Bank Accounting & Finance 16*(4), 26-30.

GNU Operating System (1996). Retrieved 25 August, 2009, from http://www.gnu.org/

Koch, G. (2007). "ABC: An Introduction to Vendor Management" Retrieved 15 February, 2010, from http://www.cio.com/article/118600/ABC_An_Introduction_to_Vendor_ Management

Lingham, L. (2009). "Vendor Management." Retrieved 17 February, 2010, from http://www.citehr.com/22963-vendor-management-system.html

Marciniak, J. J., & Reifer, J. D. (1990). *Software Acquisition Management*. New York City, NY: Wiley Series.

Nell, D., Daniel, J. T., & Chip, W. (2006). *Objected-Oriented Data Structures Using JAVA.* (Second ed.): Jones & Bartlett's Publishers.

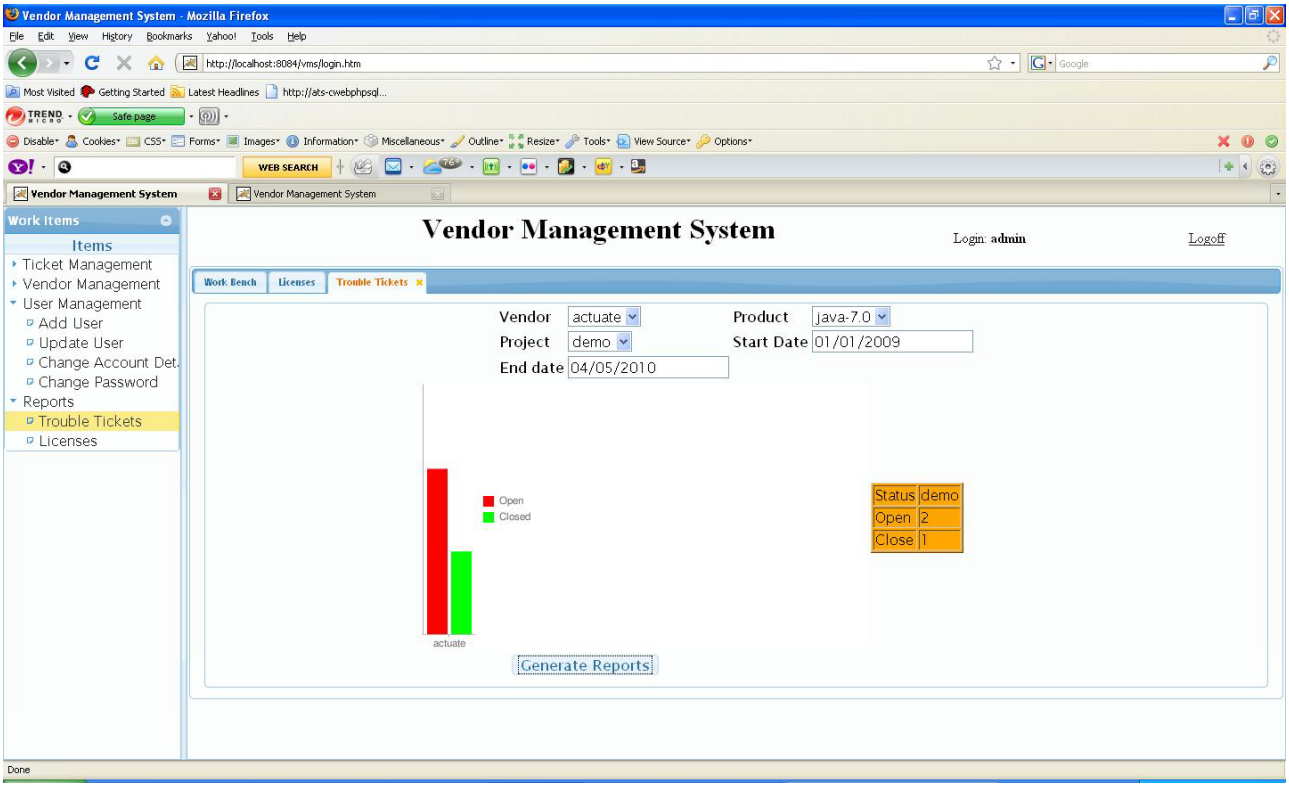
Pearlson, E. K., & Saunders, S. C. (2006). *Managing & Using Information Systems.* (Third ed.): John Wiley & Sons, Inc.

Ray, C. L. (2009). "Update Your Business With the Latest Purchasing Software: Vendor Management." *Guide to Purchasing Software: Vendor Management.* Retrieved 25 January, 2010, from http://www.business.com/directory/management/operations_management/purchasing/software/m/vendor-management/

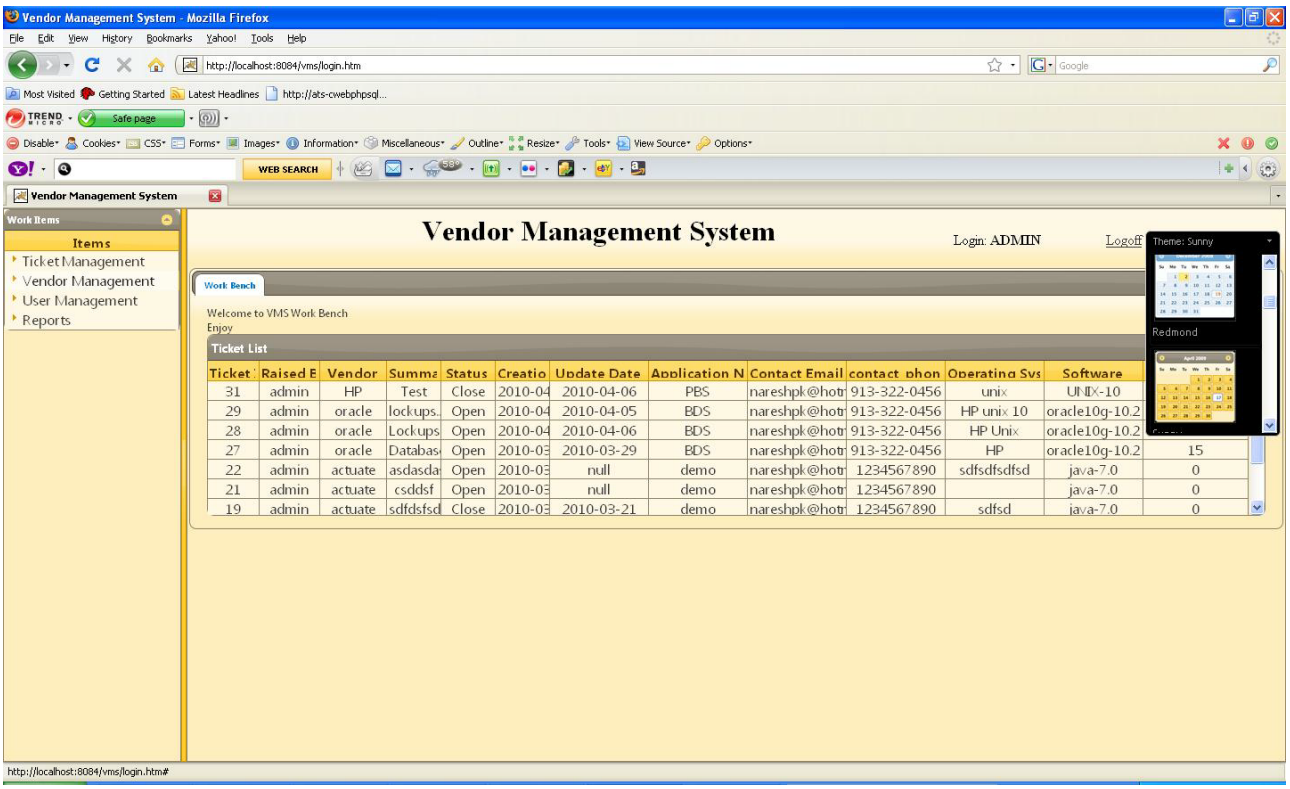
Verville, J., & Halingten, A. (2001). *Acquiring Enterprise Software: Beating the Vendors at Their Own Game.* Upper Saddle River, NJ: Prentice Hall, Inc.

Appendix A

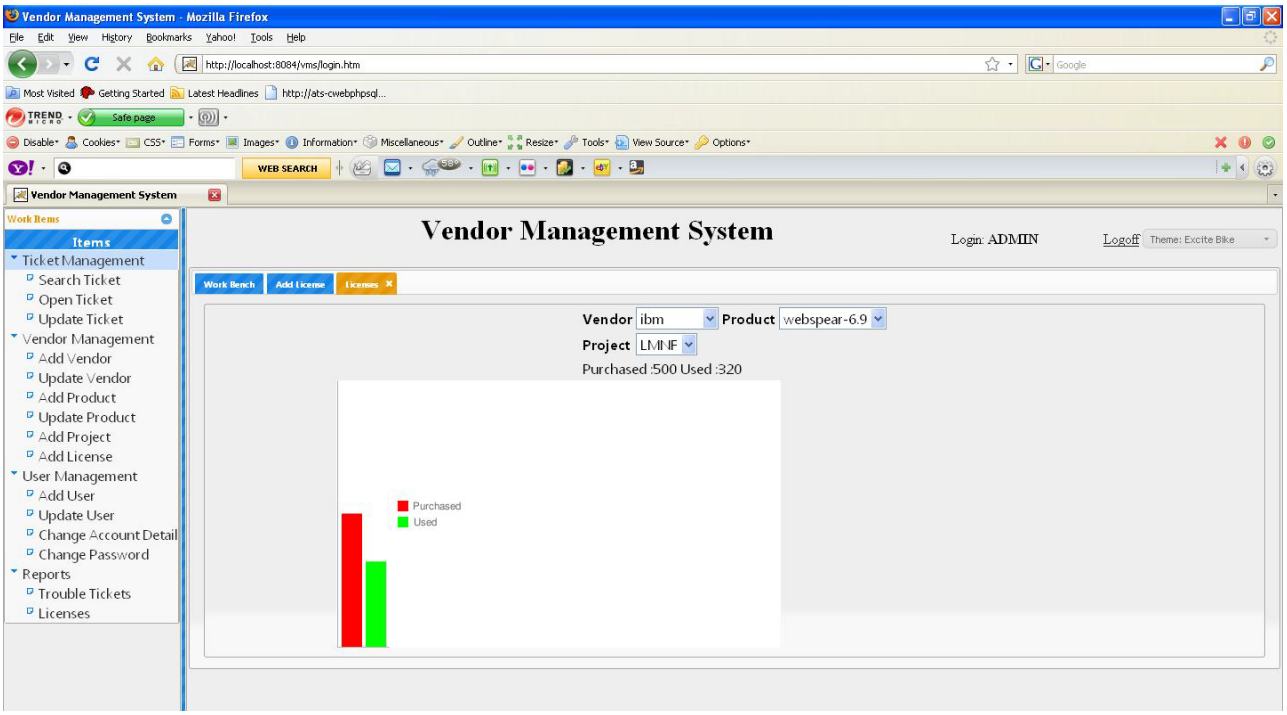
Tickets report based on vendor, product, and project



User theme selection



Vendor license report



Ticket management service class

A Java class performs search ticket, open ticket, update ticket, and forward ticket functionalities.

Method Summary	
Void	**addProduct**(Product p, java.lang.String vendor)
Void	**addProject**(java.lang.String project_name, java.lang.String project_desc, java.lang.String project_status, java.lang.String[] product_list, java.lang.String admin)
Void	**createTicket**(Ticket t, java.lang.String uname)
java.util.HashMap	**generateReport**(java.lang.String vendor, java.lang.String product, java.lang.String project, java.lang.String startdt, java.lang.String end_dt)
java.util.List	**getAllProducts**()
java.util.List	**getAllProjects**()
java.util.HashMap<java.lang.String,java.lang.Object>	**getAllTickets**(int page, int rows, java.lang.String sidx, java.lang.String sort, java.lang.String _search, java.lang.String searcFiled, java.lang.String searchOper, java.lang.String searchString)
javax.persistence.EntityManagerFactory	**getEmf**()
License	**getLicense**(java.lang.String vendor, java.lang.String product, java.lang.String project)
org.springframework.mail.MailSender	**getMailsender**()
org.springframework.mail.SimpleMailMessage	**getMessage**()
java.util.List	**getProducts**(java.lang.String vendorid)
java.util.List	**getProjects**(java.lang.String productid)

java.util.HashMap<java.lang.String,java.lang.Object>	**getVendors**()
Boolean	**isValidProduct**(ProductPK pk)
Boolean	**isValidProject**(java.lang.String project)
static void	**main**(java.lang.String[] args)
Void	**saveLicense**(License l)
Void	**setEmf**(javax.persistence.EntityManagerFactory emf)
Void	**setMailsender**(org.springframework.mail.MailSender mailsender)
Void	**setMessage**(org.springframework.mail.SimpleMailMessage message)
java.lang.String	**toSQLDate**(java.lang.String date)
Void	**updateProduct**(Product p)
Void	**updateTicket**(Ticket t, java.lang.String checked, java.lang.String role, java.lang.String userId)

User management service class

A Java class performs add a user, update user, and change user account details.

Method Summary	
void	**addUser**(User u)
void	**changePassword**(java.lang.String uname, java.lang.String pwd)
java.util.List	**getAdmins**()
java.util.HashMap<java.lang.String, java.lang.Object>	**getAllUsers**(int page, int rows, java.lang.String sidx, java.lang.String sort, java.lang.String _search, java.lang.String searcFiled, java.lang.String searchOper, java.lang.String searchString)
javax.persistence.EntityManagerFactory	**getEmf**()
User	**getUser**(java.lang.String uname)
User	**isValidUser**(java.lang.String username, java.lang.String password)
boolean	**isValidUserName**(java.lang.String uname)
static void	**main**(java.lang.String[] args)
void	**setEmf**(javax.persistence.EntityManagerFactory emf)
void	**updateUser**(User u)

License class

Hibernate entity class, which maps to license table in the VMS database.

Method Summary	
int	**getId**()
java.lang.String	**getProduct**()
java.lang.String	**getProject**()
int	**getPruchased**()
java.util.Date	**getPurchased_dt**()
java.util.Date	**getRenewed_dt**()
java.util.Date	**getUpdated_dt**()
int	**getUsed**()
java.lang.String	**getUser**()
java.lang.String	**getVendor**()
void	**setId**(int id)
void	**setProduct**(java.lang.String product)
void	**setProject**(java.lang.String project)
void	**setPruchased**(int pruchased)
void	**setPurchased_dt**(java.util.Date purchased_dt)
void	**setRenewed_dt**(java.util.Date renewed_dt)
void	**setUpdated_dt**(java.util.Date updated_dt)
void	**setUsed**(int used)
void	**setUser**(java.lang.String user)
void	**setVendor**(java.lang.String vendor)

Product class

Hibernate entity class, which maps to product table in the VMS database.

Method Summary	
boolean	**equals**(java.lang.Object object)
java.lang.String	**getContact_email**()
java.lang.String	**getContact_name**()
java.lang.String	**getContact_phone**()
java.lang.String	**getDescription**()
ProductPK	**getId**()
java.util.List<Project>	**getProjects**()
java.lang.String	**getStart_dt**()
java.lang.String	**getStatus**()
int	**hashCode**()
void	**setContact_email**(java.lang.String contact_email)
void	**setContact_name**(java.lang.String contact_name)
void	**setContact_phone**(java.lang.String contact_phone)
void	**setDescription**(java.lang.String description)
void	**setId**(ProductPK id)
void	**setProjects**(java.util.List<Project> projects)
void	**setStart_dt**(java.lang.String start_dt)
void	**setStatus**(java.lang.String status)
java.lang.String	**toString**()

Project class

Hibernate entity class, which maps to project table in the VMS database.

Method Summary	
boolean	**equals**(java.lang.Object object)
java.lang.String	**getDescription**()
java.lang.String	**getId**()
java.util.List<Product>	**getProducts**()
java.lang.String	**getStatus**()
User	**getU**()
int	**hashCode**()
void	**setDescription**(java.lang.String description)
void	**setId**(java.lang.String id)
void	**setProducts**(java.util.List<Product> products)
void	**setStatus**(java.lang.String status)
void	**setU**(User u)
java.lang.String	**toString**()

Ticket class

Hibernate entity class, which maps to ticket table in the VMS database.

Method Summary	
boolean	**equals**(java.lang.Object object)
java.lang.String	**getApp_name**()
User	**getAssaigned**()
java.util.Date	**getClose_dt**()
java.lang.String	**getContact_email**()
java.lang.String	**getContact_phone**()
java.util.Date	**getCreation_dt**()
java.lang.String	**getDescription**()
long	**getF_ticket_id**()
java.lang.Long	**getId**()
java.lang.String	**getOs**()
java.lang.String	**getSoftware**()
java.lang.String	**getStatus**()
java.lang.String	**getSummary**()
java.util.Date	**getUpdate_dt**()
User	**getUser**()
java.lang.String	**getUsers_affected**()
java.lang.String	**getVendor**()
java.lang.String	**getVersion**()
int	**hashCode**()
static void	**main**(java.lang.String[] args)
void	**setApp_name**(java.lang.String app_name)
void	**setAssaigned**(User assaigned)
void	**setClose_dt**(java.util.Date close_dt)
void	**setContact_email**(java.lang.String contact_email)
void	**setContact_phone**(java.lang.String contact_phone)
void	**setCreation_dt**(java.util.Date creation_dt)

void	**setDescription**(java.lang.String description)
void	**setF_ticket_id**(long f_ticket_id)
void	**setId**(java.lang.Long id)
void	**setOs**(java.lang.String os)
void	**setSoftware**(java.lang.String software)
void	**setStatus**(java.lang.String status)
void	**setSummary**(java.lang.String summary)
void	**setUpdate_dt**(java.util.Date update_dt)
void	**setUser**(User user)
void	**setUsers_affected**(java.lang.String users_affected)
void	**setVendor**(java.lang.String vendor)
void	**setVersion**(java.lang.String version)
java.lang.String	**toString**()

User class

Hibernate entity class, which maps to user table in the VMS database.

Method Summary	
boolean	**equals**(java.lang.Object object)
java.lang.String	**getEmail**()
java.lang.String	**getFname**()
java.lang.String	**getId**()
java.lang.String	**getLname**()
java.lang.String	**getPassword**()
java.lang.String	**getPhone**()
java.lang.String	**getRole**()
java.lang.String	**getStatus**()
java.util.List<Ticket>	**getTickets**()
int	**hashCode**()
void	**setEmail**(java.lang.String email)
void	**setFname**(java.lang.String fname)
void	**setId**(java.lang.String id)
void	**setLname**(java.lang.String lname)
void	**setPassword**(java.lang.String password)
void	**setPhone**(java.lang.String phone)
void	**setRole**(java.lang.String role)
void	**setStatus**(java.lang.String status)
void	**setTickets**(java.util.List<Ticket> tickets)
java.lang.String	**toString**()

Vendor class

Hibernate entity class, which maps to license table in the VMS database.

Method Summary	
boolean	**equals**(java.lang.Object object)
java.lang.String	**getDescription**()
java.lang.String	**getId**()
java.lang.String	**getLink**()
java.lang.String	**getName**()
java.lang.String	**getPhone**()
java.util.List<Product>	**getProducts**()
java.lang.String	**getStatus**()
int	**hashCode**()
static void	**main**(java.lang.String[] args)
void	**setDescription**(java.lang.String description)
void	**setId**(java.lang.String id)
void	**setLink**(java.lang.String link)
void	**setName**(java.lang.String name)
void	**setPhone**(java.lang.String phone)
void	**setProducts**(java.util.List<Product> products)
void	**setStatus**(java.lang.String status)
java.lang.String	**toString**()